persistance_pt1

April 1, 2021

apply “superstatistical” analysis (https://www.nature.com/articles/ncomms8516#Sec19) to walking/crawling modes

[1]:
```
# we need to address some problems with our testing of the method in␣
 ↪persistance.py
# namely, we use linearised tracks but 0.1 timestep
# probably we should use the step velocity (the velocity of each step)
```

[2]:
```
sys.path.insert(0, os.path.abspath('tools/'))
import bayesloop
import sys
import os
from copy import deepcopy
import numpy as np
import matplotlib.pyplot as plt
import plotutils
import matdef
import _fj
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-baade4547fec> in <module>
----> 1 sys.path.insert(0, os.path.abspath('tools/'))
      2 import bayesloop
      3 import sys
      4 import os
      5 from copy import deepcopy

NameError: name 'os' is not defined
```

[ ]:
```
# load fanjin data
debug = False
N = 100 if debug else None
crawling_idx, crawling_trs = _fj.slicehelper.load_linearized_trs(
    'default_crawling_list', N)
walking_idx, walking_trs = _fj.slicehelper.load_linearized_trs(
    'default_walking_list', N)
```

```
all_idx = np.concatenate([crawling_idx, walking_idx])
all_trs = crawling_trs + walking_trs
print()
print("loaded {} crawling tracks".format(crawling_idx.size))
print("loaded {} walking tracks".format(walking_idx.size))
print("total {} tracks".format(len(all_trs)))
```

[ ]:
```
# search all the tracks for some which show clear steps in aspect ratio
whaspect = [tr['length']/tr['width'] for tr in all_trs]
whaspect_std = [np.std(whaspect_i) for whaspect_i in whaspect]
a_sorted = sorted(enumerate(whaspect_std), key=lambda t: t[1], reverse=True)
print(a_sorted[:10])
n = 10
fig, axes = plt.subplots(n, 1, figsize=(10, n*5))
for i, (track_i, std) in enumerate(a_sorted[:n]):
    ax = axes[i]
    ax.set_title(r'track no. = {}, track id = {}'.format(i, track_i))
    whaspect_i = whaspect[track_i]
    ax.plot(0.1 * np.arange(whaspect_i.size),  whaspect_i)
    ax.set_xlabel('time (s)')
    ax.set_ylabel('length/width (microns)', fontsize='large')
plt.tight_layout()
plt.show()
```

[ ]:
```
# and now we need to familiarise ourselves with the code given by
# https://www.nature.com/articles/ncomms8516#Sec19
sys.path.insert(0, os.path.abspath('tools/'))
analyser = bayesloop.BayesLoop()
print('The default limits on q and a are respectively, ',
      analyser.qBound, analyser.aBound)
print(analyser.aBound)
print(analyser.qBound)
# where the limits on q are expected to be [-1, 1] so its not entirely clear␣
 ↪why [-1.5,1.5] is used
print('Default control parameters.')
print('pmin = {}'.format(analyser.pMin))
print('Box kernel halfwidths (Ra, Rq) = ({}, {})'.format(analyser.Ra, analyser.
 ↪Rq))
print('default gridsize =', analyser.gridSize)
print('kernal size is in the context of gridsize and the limits so in fact␣
 ↪kernel dimensions are ({},{})'.format(
    2 * analyser.Ra * (analyser.aBound[1] -
                      analyser.aBound[0])/analyser.gridSize,
    2 * analyser.Rq * (analyser.qBound[1]-analyser.qBound[0])/analyser.gridSize
))
```

[ ]:
```python
# lets pick track #2 to work with because it appears to switch to walking and␣
↪back
sorted_pick_id = 2
eye_track_id, eye_track_std = a_sorted[sorted_pick_id]
eye_data_id = all_idx[eye_track_id]
print('track data id =', eye_data_id)
eye_track = all_trs[eye_track_id]
```

[ ]:
```python
# And again for reference, plot apsect ratio
ax = plt.gca()
eye_track_aspect = whaspect[eye_track_id]
ax.plot(0.1 * np.arange(eye_track_aspect.size),  eye_track_aspect)
ax.set_xlabel('time (s)')
ax.set_ylabel('length/width (microns)')
plt.show()
```

[ ]:
```python
# and velocity so that we know if the bacterium stops moving
```

[ ]:
```python
# xy data
trackxy = np.column_stack([eye_track['x'], eye_track['y']])
print('check xy data shape ', trackxy.shape)
print('write out this data so we can check it against the GUI tool')
target = 'tools/trackxy_{:04d}.dat'.format(eye_data_id)
print('writing track {:04d} to {}'.format(eye_data_id, target))
np.savetxt(target, trackxy)
```

[ ]:
```python
# setup and run the analysis
# analysis takes velocity data
track_u = (trackxy[1:] - trackxy[:-1])/matdef.TIMESTEP
speed = np.linalg.norm(track_u, axis=1)
print('velocity lims ({},{})'.format(speed.min(), speed.max()))
analyser.data = track_u
analyser.pMin = 1e-18  # see persistance.py
analyser.startAnalysis()
```

[ ]:
```python
#
# mean parameters postMean
print(analyser.postMean.shape)
print(analyser.avgPost.shape)
# rescale eye_track_aspect to [0,1]
rescale = 1/np.quantile(eye_track_aspect, 0.95)
aspect_ghost = rescale * eye_track_aspect
kwghost = {'linewidth': 4, 'color': '0.2', 'alpha': 0.4}
linekw = {'linewidth': 4}


def plot_qa(axes, analyser, aspect_ghost=aspect_ghost):
    basis = np.arange(analyser.postMean[0].size)
    _plot_qa(axes, basis, analyser.postMean, analyser.pMin,␣
 ↪aspect_ghost=aspect_ghost)

def _plot_qa(axes, basis, postMean, pMin, aspect_ghost=aspect_ghost,␣
 ↪xlabel='timestep'):
    ax1, ax2 = axes
    q_mean, a_mean = postMean
    # ax1.axhline(rescale, linewidth=1, c='k', alpha=0.6, linestyle='--')
    ax2.set_xlabel(xlabel)
    if not (aspect_ghost is None):
        aspect_ghost = aspect_ghost[1:-1] # same shape as q,a
        ax1.plot(basis, aspect_ghost, label='rescaled length/width', **kwghost)
        ax2.plot(basis, aspect_ghost, **kwghost)
    ax1.set_title('pmin = {:.2E}'.format(pMin))
    ax1.set_ylim((0, 1.5))
    ax1.plot(basis, q_mean, **linekw)
    ax1.set_ylabel(r'$q_t$')
    ax2.set_ylim(ymin=0.0)
    ax2.plot(basis, a_mean, label='a parameter', **linekw)
    ax2.set_ylabel(r'$a_t$')
    ax1.legend()
    ax2.set_ylim((0, 1.5))
```

[ ]:
```
#

fig, axes = plt.subplots(2, 1, figsize=(10, 2*5))
plot_qa(axes, analyser)
plt.show()
```

[ ]:
```
# check box kernel
side_analyser = bayesloop.BayesLoop()
side_analyser.pMin = 1e-18
side_analyser.Ra = 0
side_analyser.Rq = 0
side_analyser.data = track_u
side_analyser.kernel_on = False
side_analyser.startAnalysis()
del side_analyser.postSequ
```

[ ]:
```
print(len(eye_track.step_idx))
fig, axes = plt.subplots(2, 1, figsize=(10, 2*5))
plot_qa(axes, side_analyser)
ax1, ax2 = axes
# for i in eye_track.step_idx:
#     ax1.axvline(i, alpha=0.5)
plt.show()
```

At this point we make an important observation. The q,a values are not supposed to be stuck on particular values so easily. I suspect this is because we are doing something which is conceptually bad by using linearised tracks but 0.1 timestep.

```
[ ]: # first we can check this by doing the same analysis on the un-linearised data
     tr_unlinearised = _fj.trackload([eye_data_id])[0]
     xy = np.column_stack([tr_unlinearised['x'], tr_unlinearised['y']])
     tr_u = (xy[1:] - xy[:-1])/matdef.TIMESTEP
     analyser.data = tr_u
     analyser.startAnalysis()
     del analyser.postSequ
```

```
[ ]: # plotting^^
     fig, axes = plt.subplots(2, 1, figsize=(10, 2*5))
     plot_qa(axes, analyser, aspect_ghost=None)
     plt.show()

     # # %%
     # # using analyser with box kernel as well for good measure
     # analyser.data = tr_u
     # analyser.startAnalysis()
     # del analyser.postSequ
     # fig, axes = plt.subplots(2, 1, figsize=(10, 2*5))
     # plot_qa(axes, analyser)
     # plt.show()
     # # which appears to be idenitcal
```

no change!? so our guess was wrong, this result has nothing to do with our linearisation… even so treating each step in the linearisation as independent measurements makes more sense so we still need to try that

```
[ ]: # computing velocity of steps
     step_idx = np.array(eye_track.step_idx)
     step_xy = np.column_stack([eye_track['x'][step_idx], eye_track['y'][step_idx]])
     step_time = matdef.TIMESTEP * (step_idx[1:] - step_idx[:-1])
     time_basis = eye_track['time'][step_idx] - eye_track['time'][0]
     step_u = (step_xy[1:] - step_xy[:-1]) / step_time[:, np.newaxis]
     step_aspect = whaspect[eye_track_id][step_idx]
     step_aspect_ghost = 1/np.quantile(step_aspect, 0.95) * step_aspect
     print(track_u.shape, step_u.shape)
     # so we went from ~2000 down to ~200 data points
```

```
[ ]: analyser.data = step_u
     analyser.pMin = 1e-18
     analyser.startAnalysis()
     del analyser.postSequ
     # check sizes
     def check_sizes():
```

```
    print(time_basis.shape)
    print(analyser.postMean.shape)
    print(step_aspect_ghost.shape)
check_sizes()
```

[ ]:
```
fig, axes = plt.subplots(2, 1, figsize=(10, 2*5))
_plot_qa(axes, time_basis[1:-1], analyser.postMean, analyser.pMin,␣
 ↪aspect_ghost=step_aspect_ghost,
    xlabel='time (s)')
plt.show()
```

[ ]:
```
# and if for some reason we refuse to use box kernel
side_analyser.data = step_u
side_analyser.startAnalysis()
del side_analyser.postSequ
fig, axes = plt.subplots(2, 1, figsize=(10, 2*5))
plot_qa(axes, side_analyser, aspect_ghost=step_aspect_ghost)
plt.show()
```

[ ]:
```
# but we coarse grained out data so we definitely need to check again that we␣
 ↪used appropriate pmin

pmin_try = [1e-5, 1e-7, 1e-9, 1e-12, 1e-15, 1e-18, 1e-21]
analyser.data = step_u
save_analyser = []
for pmin in pmin_try:
    print('setting pMin = {} ...'.format(pmin))
    analyser.pMin = pmin
    analyser.startAnalysis()
    del analyser.postSequ
    save_analyser.append(deepcopy(analyser))
```

[ ]:
```
# plotting goes in a new cell

n = len(save_analyser)
fig, axes = plt.subplots(2*n, 1, figsize=(10, n*2*5))
for i, analyser in enumerate(save_analyser):
    ax_pair = (axes[2*i], axes[2*i+1])
    plot_qa(ax_pair, analyser, aspect_ghost=step_aspect_ghost)
plt.tight_layout()
plt.show()
```

We see that pmin = 1e-18 is still a good choice and also that when using step velocities the method appears to find that persistence in the first part of the trajectory is actually less than that in the last part. which I tentativly suggest is higher specificity due to not spamming the algorithm with velocities which are not statistically independent events.

[ ]:

```python
# as a sanity check we attempt to break the analysis by cutting it at timestep
check_analyser = bayesloop.BayesLoop()
check_analyser.pMin = 1e-18
# get index of time 50
cut_idx = np.searchsorted(time_basis, 50)
print('cutting at index ', cut_idx)
cut_time_basis = time_basis[cut_idx:]
cut_step_u = step_u[cut_idx:]
# check sizes again
print(cut_time_basis.shape, cut_step_u.shape)
check_analyser.data = cut_step_u
check_analyser.startAnalysis()
```

[ ]:

```python
# and plot this one
cut_step_aspect_ghost = step_aspect_ghost[cut_idx:]
fig, axes = plt.subplots(2, 1, figsize=(10, 2*5))
_plot_qa(axes, cut_time_basis[1:-1], check_analyser.postMean,
    check_analyser.pMin, aspect_ghost=cut_step_aspect_ghost,
    xlabel='time (s)')
plt.show()
```

check what Fanjin has to say about the precision of their velocity > We estimate this noise to be approximately 0.03 m (0.5 pixels), which sets a lower threshold of approximately 0.3 m⁄s on the velocities that can be resolved given the imaging rate of 10 frames⁄sec. To accurately extract velocities from the trajectory that fall below this noise threshold, we first subdivide the trajectory into segments using a noise threshold of 2 pixels (0.12 m). If the position varies less than this threshold, we calculate the velocity using linear regression across the entire segment

Ok so after much testing it seems useful to keep the box kernel and pmin=1e-18 and also it is sensible to use the step velocity so we can task the cluster with computing that one as well …

[0]:

```
```

[0]:

```
```